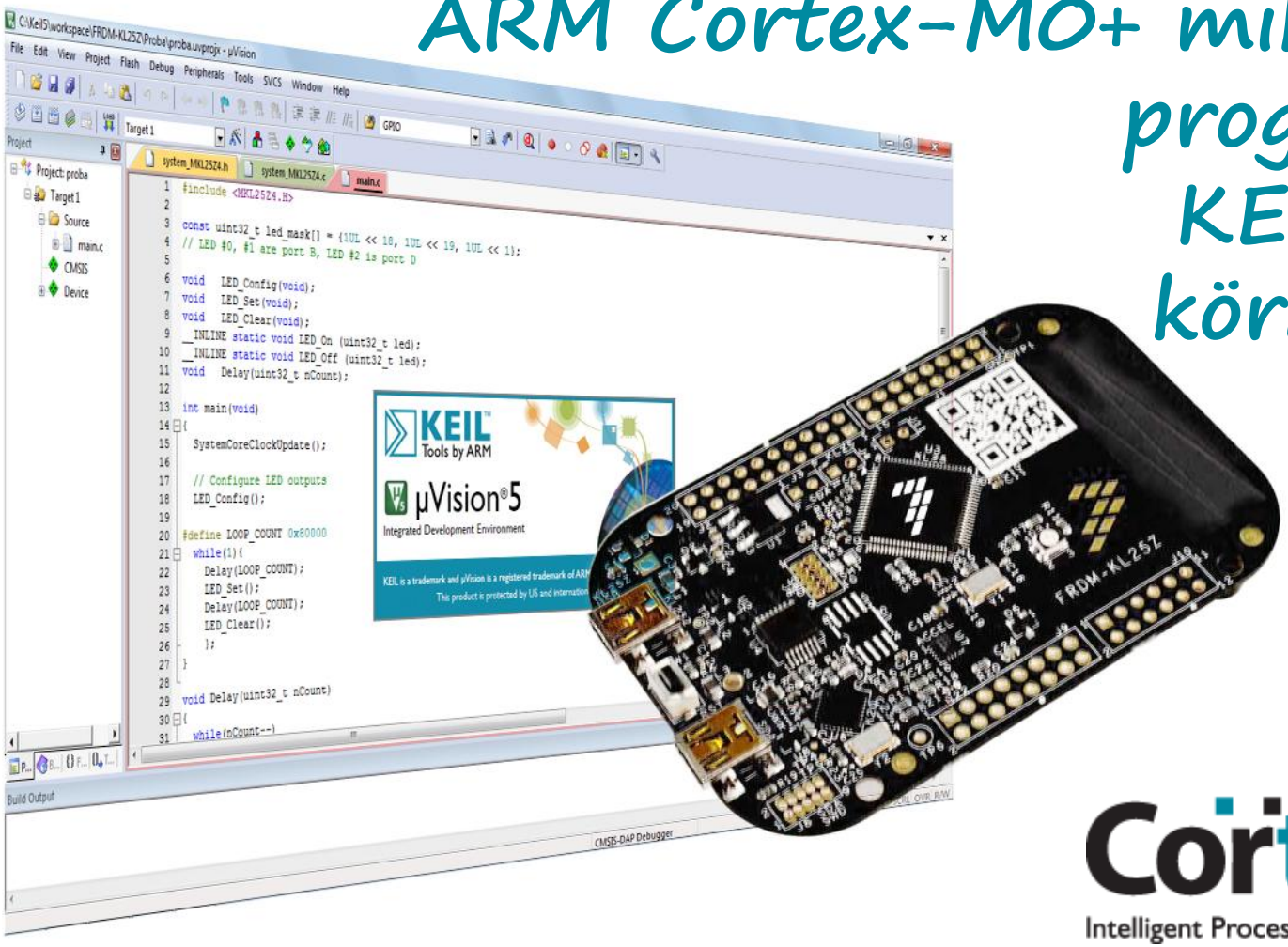
ARM Cortex-M0+ mikrovezérlő programozása KEIL MDK 5 környezetben



Cortex
Intelligent Processors by ARM

8. Az SPI kommunikációs csatorna

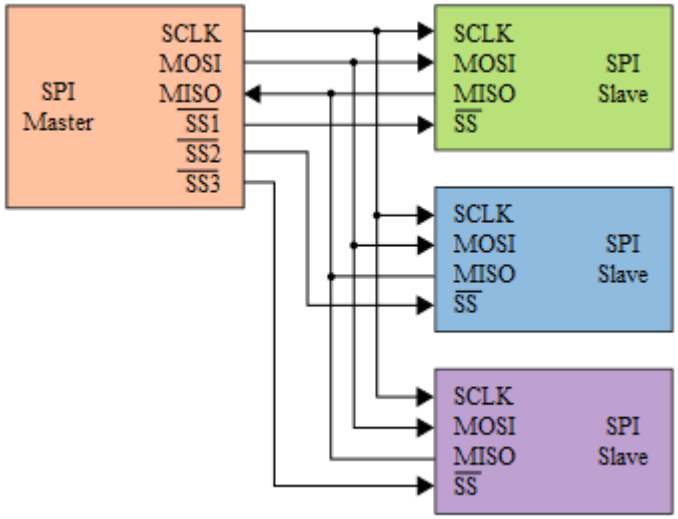
Felhasznált anyagok, ajánlott irodalom

- Joseph Yiu: **The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors** (2nd Ed.)
- Muhammad Ali Mazidi, Shujen Chen, Sarmad Naimi, Sepehr Naimi: **Freescale ARM Cortex-M Embedded Programming**
- ARM University Program: **Course/Lab Material for Teaching Embedded Systems/MCUs** (for the **FRDM-KL25Z** board)
- ARM Information Center: **Cortex-M0+ Devices Generic User Guide**
- Freescale: **MKL25Z128VLK4 MCU datasheet**
- Freescale: **KL25 Sub-Family Reference Manual**
- Freescale: **FRDM-KL25Z User Manual**
- Freescale: **FRDM-KL25Z Pinouts**

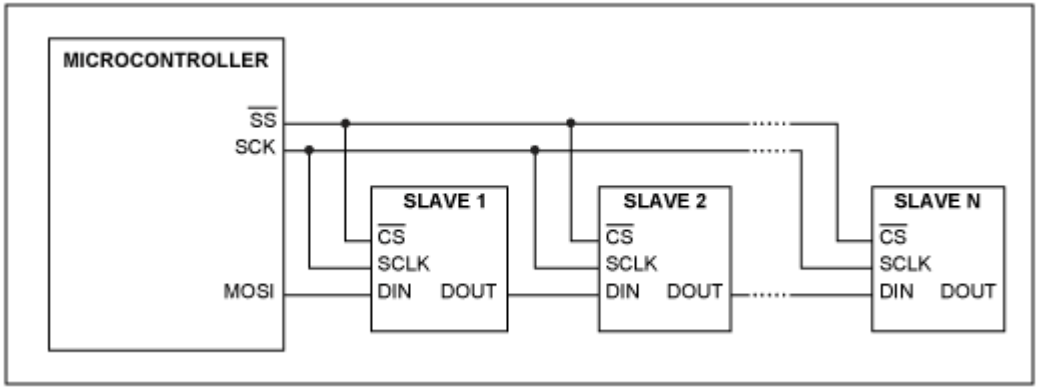
Soros Periféria Illesztő

Az **SPI** (Serial Peripheral Interface) busz kétirányú, szinkron soros kommunikációt valósít meg két eszköz között, amelyek master/slave (mester/szolga) viszonyban állnak.

Az **SPI** busz kiterjeszthető: egy master több slave eszközhöz is kapcsolódhat, ám a kommunikációra kiválasztott slave eszközt egyedi választó vonallal (*Slave Select*) hardveresen kell kijelölni.



A másik lehetőség az SPI eszközök felfűzése (daisy chain) – amennyiben az eszközök ezt a módot támogatják.



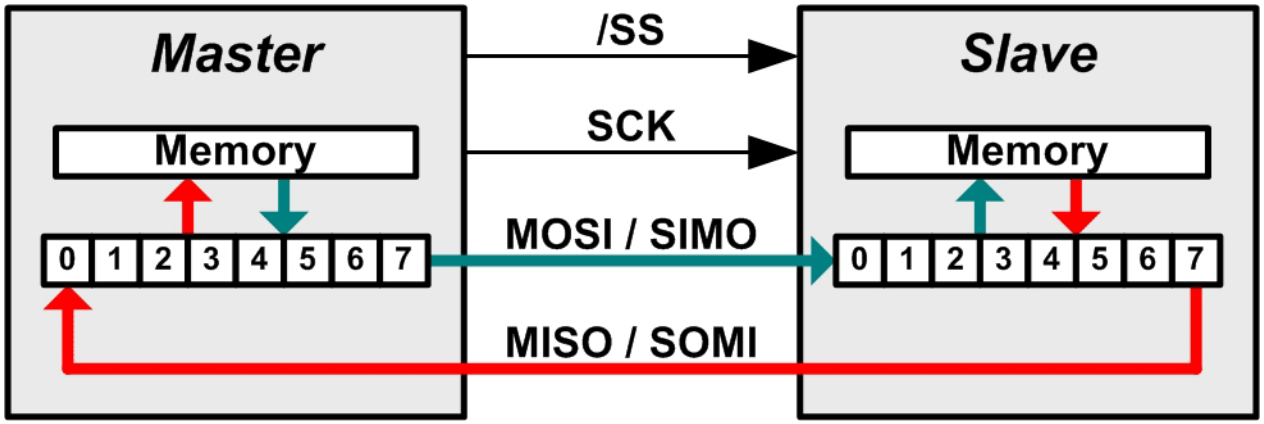
Az SPI busz jelei

SCK - Az SPI busz szinkronizálást biztosító órajele. A master eszköz állítja elő.

$\overline{SS}$ - Slave select, azaz a slave eszköz kiválasztására szolgáló jel, melynek '0' állapota aktivizál. A master eszköz állítja elő. Slave eszköz esetében az SPI modul speciális kivezetését kell használnunk az eszköz kiválasztásához, de Master eszköznél elvileg bármilyen GPIO lábat használhatunk a slave eszköz megszólítására.

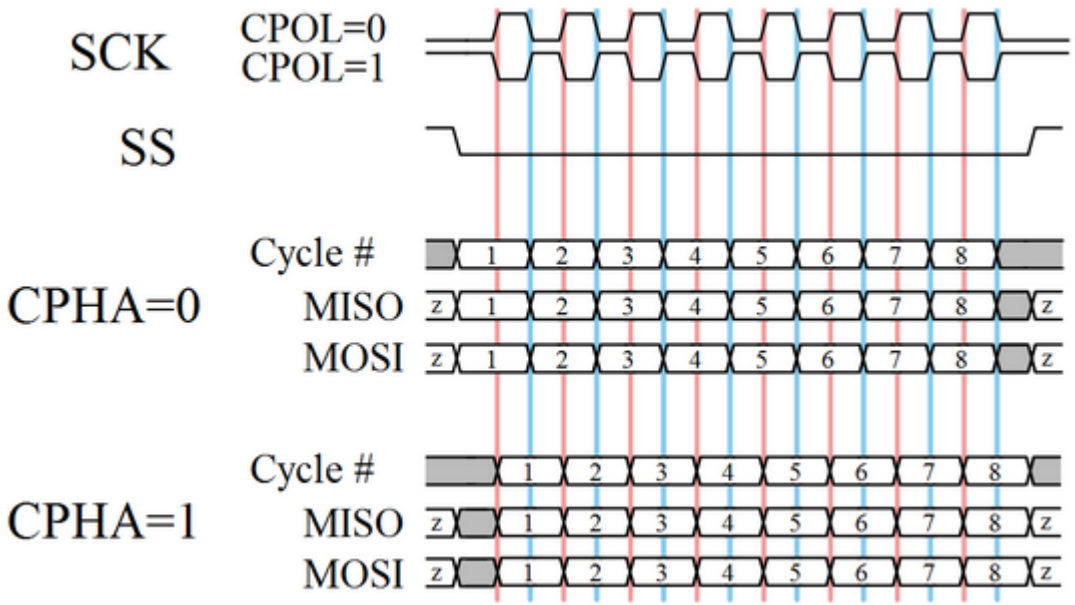
MOSI - A master eszköz kimeneti adatvonala (Master out, Slave in). A master eszköz állítja elő.

MISO - A slave eszköz kimeneti adatvonala, melyet a master olvas (Master in, Slave out). A slave eszköz állítja elő.



Ábra forrása: http://www.eetimes.com/document.asp?doc_id=1272534

Az órajel polaritása és fázisa



SPI mód	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

A lehetséges üzemmódok az órajel polaritásának és fázisának kombinációjaként állnak elő.

Az órajel polaritása:
CPOL = 0 esetén **SCK** inaktív állapota alacsony szint, **CPOL = 1** esetén magas szint.

Az órajel fázisa: **CPHA = 0** esetén az adatvonalak bekapuzása az órajel páratlan számú átmenetein történik (az ábrán rózsaszín vonalak), **CPHA = 1** esetén pedig a páros számú átmeneteknél (kék vonalak)

A FRDM-KL25Z kártya SPI perifériái

Az **MKL25Z128VLK4** mikrovezérlő két **SPI** modult tartalmaz (**SPI0** és **SPI1**)

- Az **SPI** modulok 8 bites adatformátumot támogatnak
- Az **SPI0** modul órajel forrása a busz órajel, az **SPI1** modulé pedig a CPU órajel.
- Az SPI modulok támogatják a DMA adatátvitelt és VLPS energiatakarékos módban is működnek (slave módban). Ébresztésre is képesek, amikor adatot kapnak.
- Master vagy slave módú működés választható
- Full-duplex vagy egyvezetékes kétirányú mód
- Programozható adatküldési sebesség (bitráta)
- Kettős pufferelésű adó és vevő adatregiszterek
- Órajel polaritása és fázisa konfigurálható (mind a négy mód beállítható)
- Automatikusan kiküldött "Slave select" jel (opcionális)
- Hibás mód jelzés fogadása és megszakításkérés
- Választható bitsorrend: MSB-first vagy LSB-first kiléptetési sorrend
- Vett adat egyezésének hardveres figyelése előre megadott értékkel

Az SPI modulokhoz rendelhető kivezetések

Az **SPI0** és **SPI1** modulokhoz az alábbi portkivezetéseket rendelhetjük hozzá:

SPI modul	SCLK	MOSI/MISO	SS
SPI0	PTA15, PTC5, **PTD1**	PTA16, PTA17, PTC6, PTC7, **PTD2, PTD3**	PTC4, **PTD0**
SPI1	PTB11, PTD5, PTE2	PTB16, PTB17, PTD6, PTD7, PTE1, PTE3	PTB10, PTD4, PTE4

Az **Arduino kompatibilis** bekötéshez a táblázatban kövéren szedett kivezetések tartoznak:

- **SS = PTD0**
- **SCLK = PTD1**
- **MOSI = PTD2**
- **MISO = PTD3**

A **FRDM-KL25Z Pinouts** dokumentumban található, hogy az adott funkció kiválasztásához milyen kódot kell megadni a megfelelő portvezérlő regiszter **MUX** bitcsoportjába.

Például:

```
SIM->SCGC5 |= 0x1000;   // Port D engedélyezése
PORTD->PCR[1] = 0x200; // PTD1 legyen SPI SCLK
PORTD->PCR[2] = 0x200; // PTD2 legyen SPI MOSI
```

KL25Z soros perifériák engedélyezése

SIM_SCGC4 regiszter

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1				0				SPI1	SPI0	0		CMP	USBOTG	0	
W																
Reset	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0		0	UART2	UART1	UART0	0		I2C1	I2C0	1		0			
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0

- A használni kívánt perifériát engedélyezni kell a **SIM_SCGC4** regiszter megfelelő bitjének 1-be állításával

Az SPI modulok regiszterkészlete

Az **SPI** modulok azonos regiszterkészlettel rendelkeznek. Az alábbi táblázatban az **SPI*x*** regisztereknek (ahol *x* = 0, vagy 1) csak a báziscímekhez képesti eltolási címét (ofszet cím) adtuk meg.

SPI modul	Báziscím
SPI0	0x4007 6000
SPI1	0x4007 7000

Ofszet cím	Regiszter neve, funkciója	Méret	Elérés	Reset
0x0000	**SPIx_C1** control register (1. vezérlő regiszter)	8 bit	R/W	0x04
0x0001	**SPIx_C2** control register (2. vezérlő regiszter)	8 bit	R/W	0x00
0x0002	**SPIx_BR** baud rate register (adatsebesség regiszter)	8 bit	R/W	0x00
0x0003	**SPIx_S** status register (állapotjelző regiszter)	8 bit	R	0x20
0x0005	**SPIx_D** data register (adatregiszter)	8 bit	R/W	0x00
0x0007	**SPIx_M** match register (adategyezés regiszter)	8 bit	R/W	0x00

Az SPIx_C1 vezérlő regiszter

Bit	7	6	5	4	3	2	1	0
Read Write	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset	0	0	0	0	0	1	0	0

A megjelölt bitek beállításával a legegyszerűbb használatnál is foglalkoznunk kell!

SPIE - Megszakítás engedélyezése a vételi puffer megtelése (**SPRF**) illetve üzemmód hiba (**MODF**) esetén (**0:** megszakítás tiltás, **1:** megszakítás engedélyezés)
SPE - Az SPI modul működésének engedélyezése (**0:** tiltás, **1:** engedélyezés)
SPTIE - Megszakítás engedélyezése ha az SPI adatküldő puffere üres (**SPTEF**) eseménykor (**0:** megszakítás tiltás - lekérdezéses mód **1:** megszakítás engedélyezés)
MSTR - Master / Slave mód választása (**0:** slave mód, **1:** master mód)
CPOL - Órajel polaritása (**0:** nyugalmi szint alacsony, **1:** nyugalmi szint magas)
CPHA - Órajel fázisa (**0:** első átmenet eltolással, **1:** első átmenet eltolás nélkül
SSOE - Csak master módban és a MODFEN = '1' esetén hatásos (**0:** az **SS** kivezetés **MODFAULT** bemenet, **1:** az **SS** kivezetés automatikusan kezelt **SS** kimenet).
LSBFE - Bitsorrend választása (**0:** a legmagasabb helyiértékű bit megy ki először - **MSB-first** mód, **1:** a legalacsonyabb helyiértékű bit megy ki először - **LSB-first** mód)

Az SPIx_C2 vezérlő regiszter

Bit	7	6	5	4	3	2	1	0
Read Write	SPMIE	Reserved	TXDMAE	MODFEN	BIDIROE	RXDMAE	SPISWAI	SPC0
Reset	0	0	0	0	0	0	0	0

SPMIE - Megszakítás engedélyezése adategyezés esetén (**0:** megszakítás tiltás, **1:** megszakítás engedélyezés)

TXDMAE - DMA adatküldés engedélyezése (**0:** tiltás, **1:** engedélyezés)

MODFEN - Master módban "módbeállítás hiba" fogadásának engedélyezése

BIDIROE - Kétirányú mód esetén (amikor SPC0 = 1) az adatáramlás irányát állítja be (0: bemenet, 1: kimenet)

RXDMAE - Adatfogadás DMA támogatással. Ha ez a bit '1', akkor DMA átvitel történik, ha SPRF és SPE egyaránt '1'.

SPISWAI - SPI leállítása Stop módban (0: az SPI WAIT módban is működik, 1: az SPI leáll, amikor az MCU leáll)

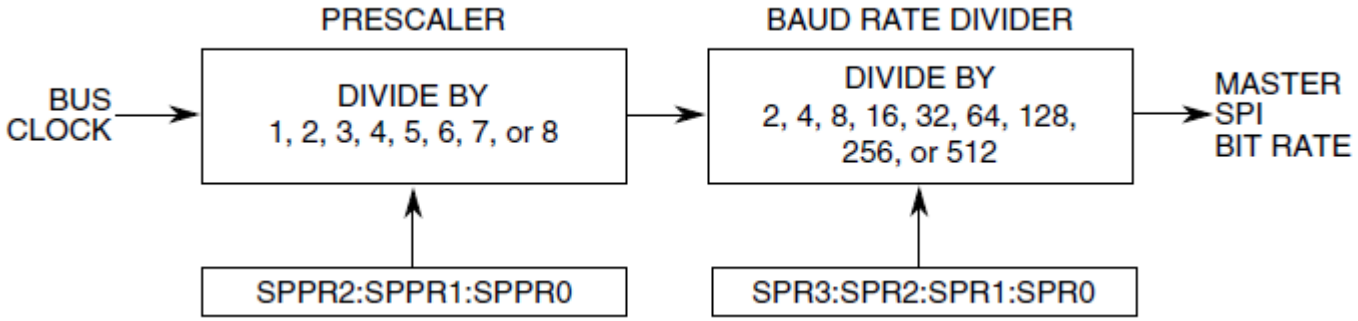
SPCO - Kétirányú adatvonal engedélyezése (0: szétválasztott adatvonalak, 1: közös, kétirányú adatvonal)

Az SPIx_BR regiszter

Ebben az adatsebességet konfigurálhatjuk egy előosztó és egy osztó beállításával.

Bit	7	6	5	4	3	2	1	0
Read	0	SPPR[2:0]			SPR[3:0]			
Write								
Reset	0	0	0	0	0	0	0	0

SPPR - Előosztási arány (0 - 7 közötti értéket írhatunk bele, a leosztás **SPPR + 1** lesz)
SPR - Frekvenciaosztási arányt írhatunk bele (csak a 0 - 8 közötti értékek érvényesek!)



Az adatsebesség: $f_{SPI} = f_{IN} / ((SPPR + 1) * 2^{(SPR+1)})$, ahol f_{IN} az **SPI0** modul esetén a buszfrekvenciát, **SPI1** esetén pedig a CPU frekvenciát jelenti. Például 48 MHz-es CPU frekvencia és 24 MHz-es buszfrekvencia esetén az 0x53 érték beírása az alábbi adatsebességeket eredményezi:

```
SPI0->BR = 0x53; // fspi = 24 MHz / (6 * 2^4) = 24 MHz/ (6*16) = 0.25 MHz = 250 kHz
SPI1->BR = 0x53; // fspi = 48 MHz / (6 * 2^4) = 48 MHz/ (6*16) = 0.50 MHz = 500 kHz
```

Az SPIx_S állapotjelző regiszter

Bit	7	6	5	4	3	2	1	0
Read	SPRF	SPMF	SPTEF	MODF	0			
Write								
Reset	0	0	1	0	0	0	0	0

SPRF - SPI vételi puffer megtelt (**0:** nincs kiolvasni való adat, **1:** adatbeolvasás befejeződött, a vett adat kiolvasható). A jelzőbit automatikusan törlődik, ha olvassuk az **SPRF** jelzőbitet, majd kiolvassuk az **SPIx_D** adatregisztert.

SPMF - Adategyezés jelzése. Ez a bit akkor áll '1'-be, ha beérkezett egy bájt (**SPRF** = 1) és a beérkezett adat megegyezik azzal, amit az **SPIx_M** match regiszterbe írtunk. (**0:** nincs egyezés, **1:** egyezés van)

SPTEF - Az SPI adatküldő puffere üres, újabb írásra kész (**0:** az adatküldő regiszter foglalt **1:** az adatregiszter felszabadult)

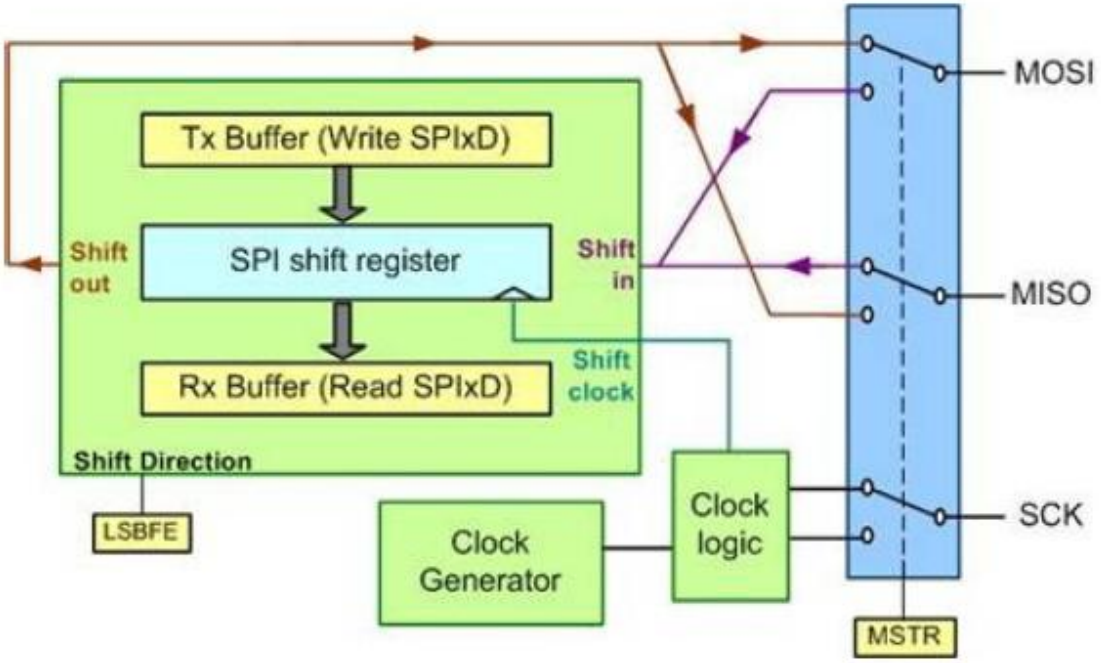
MODF - Üzemmód beállítási hibajelzés érkezésének jelzése (**0:** nem érkezett hibajelzés, **1:** hibajelzés érkezett)

A master módú tranzakció végét nem **SPTEF,** hanem **SPRF** jelzi, ezt célszerű figyelni!

Az SPIx_D adatregiszter

Az adatregiszter írásakor és olvasásakor valójában fizikailag két, különálló regisztert kezelünk. Küldéskor ide írjuk a kiküldendő bájtot, olvasáskor pedig ezen a címen olvashatjuk ki a beérkezett adatot.

Az adat kiolvasása egyúttal automatikusan törli az **SPRF** jelzőbitet is.



Az SPIx_M adategyezés regiszter

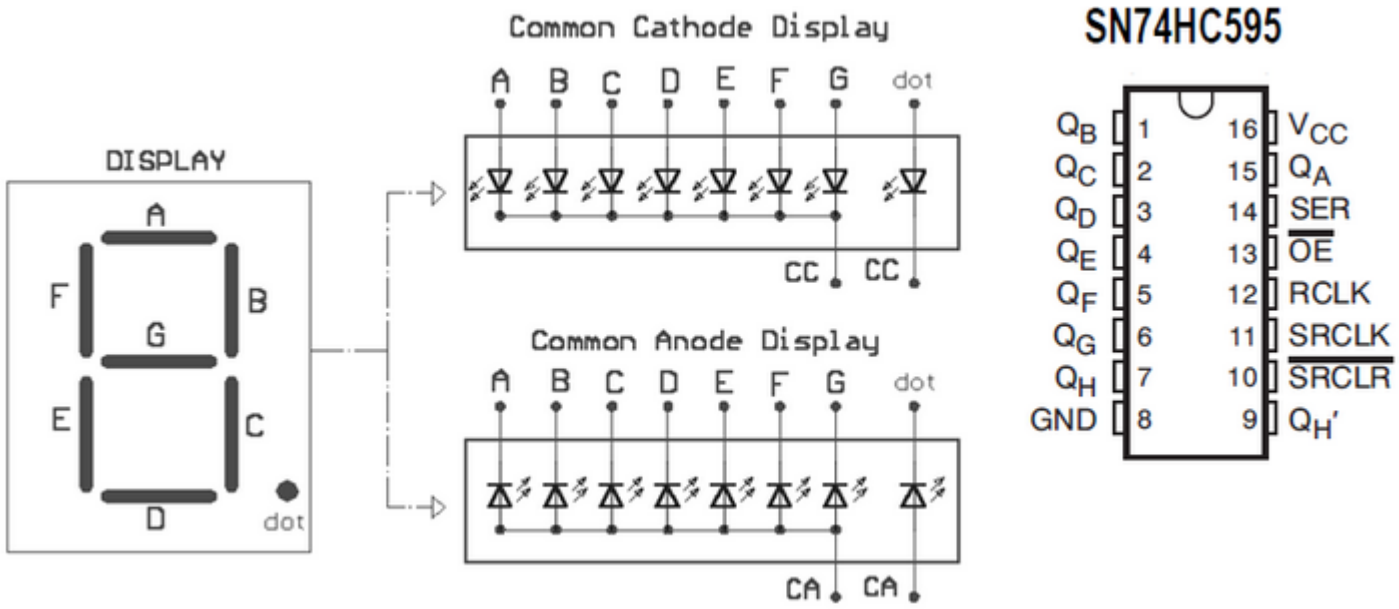
Ebbe a regiszterbe írhatjuk bele azt a számot, ami a hardveres adategyezés figyelésének az alapja. Ha az **SPI** modul által vett adat megegyezik az itt beállított értékkel, akkor az **SPIx_S** állapotjelző regiszter **SPMF** bitje '1'-be áll. Az **SPIx_C2** vezérlőregiszter **SPMIE** bitjének '1'-be állításával megszakítást is engedélyezhetünk az adategyezési eseményhez.

Mintaprogramok

- **Program8_1:** SPI0 adatküldés
- **Program8_2:** Számkijelző vezérlése MAX7219 IC-vel
- **Program8_3:** 8x8 LED mátrix vezérlése MAX7219 IC-vel

Program8_1: SPI0 adatküldés

Bemutatjuk az **SPI0** csatorna konfigurálását és az egyszerű adatküldést. A kiküldött adatokkal egy kijelzőt vezérlünk, ami két **74HC595** shift regiszterből (léptetőregiszter) és két **HD1131R** típusú 7 szegmenses LED számkijelzőből áll.

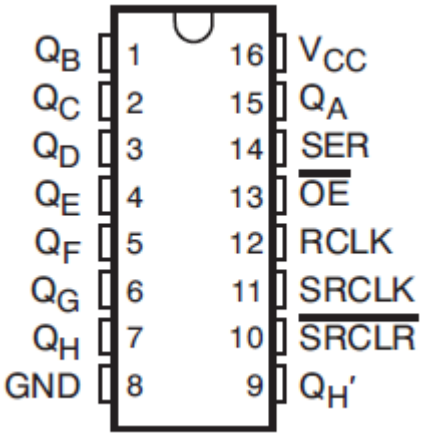


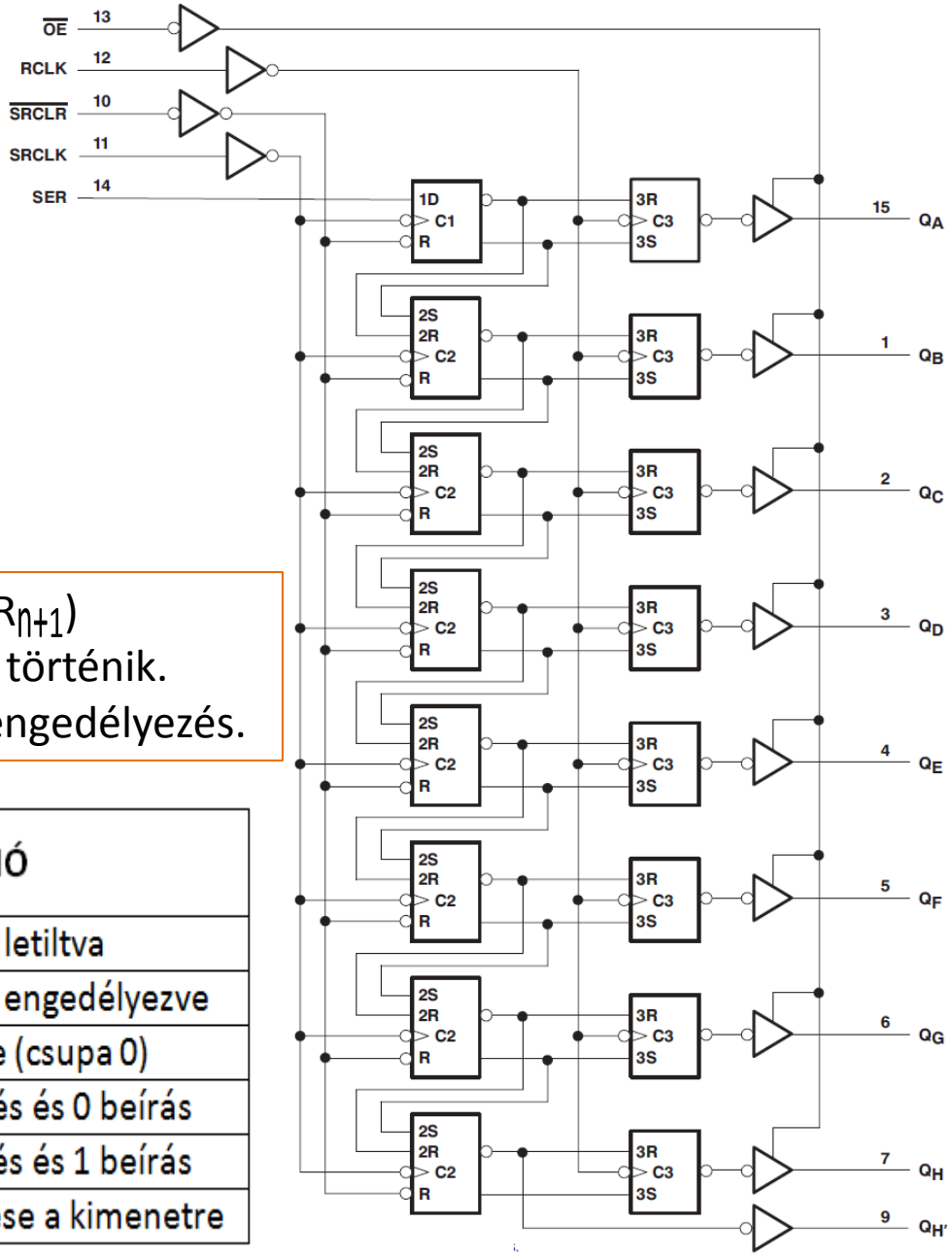
A **74HC595** shift regiszter soros bemenete a 14. láb (**SER**), kimenete pedig a 9. láb (**QH***).

A shift regiszter léptetését az **SCK** jel végzi.

Az **SS** jelet az **RCK** bemenetekre kötöttünk rá, ami az átvitel végén (felfutó él) a kimeneti adattároló regiszterébe kapuzza át az adatokat (a küldött adatbitek csak ekkor jelennek meg QA..QH kimeneteken).

SN74HC595



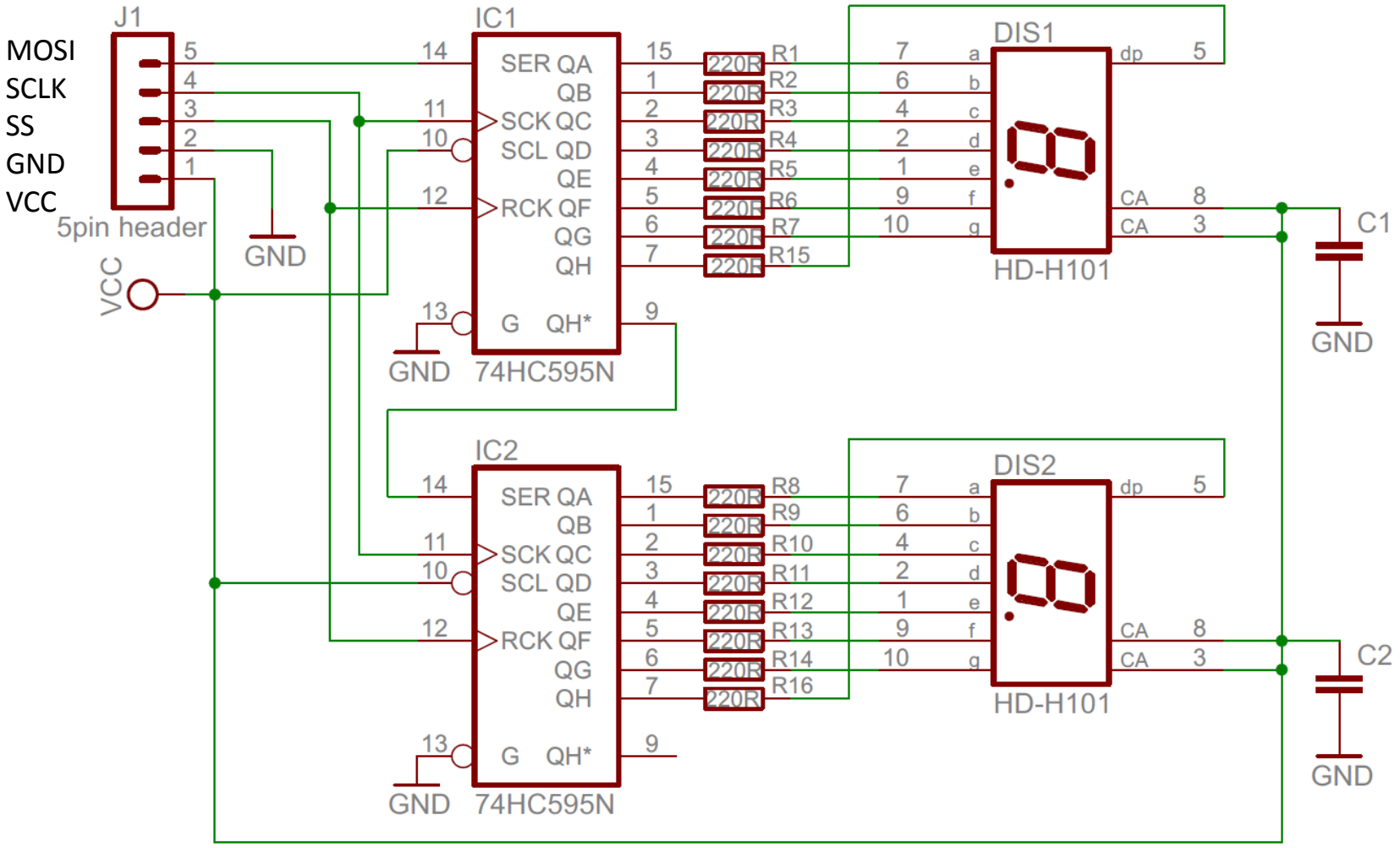


A **74HC595** IC-k sorbaköthetőek ($QH'_{n} \rightarrow SER_{n+1}$)
Az adatok áttöltése az **RCLK** jel felfutó élénél történik.
SRCLR = shift regiszter törlés, **OE** = kimenet engedélyezés.

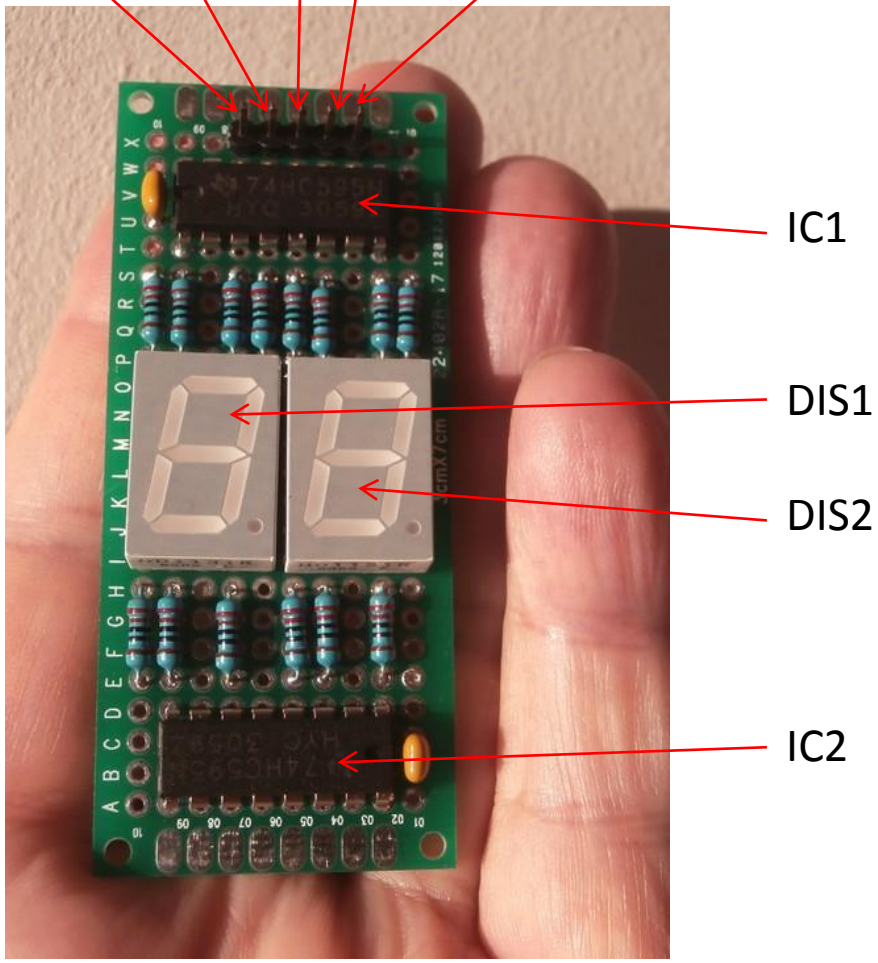
BEMENETEK					FUNKCIÓ
SER	SCLK	SCLR	RCLK	OE	
X	X	X	X	H	A Q_A – Q_H kimenetek letiltva
X	X	X	X	L	A Q_A – Q_H kimenetek engedélyezve
X	X	L	X	X	Shift regiszter törlése (csupa 0)
L	↑	H	X	X	Shift regiszter léptetés és 0 beírás
H	↑	H	X	X	Shift regiszter léptetés és 1 beírás
X	X	X	↑	X	Shift regiszter áttöltése a kimenetre

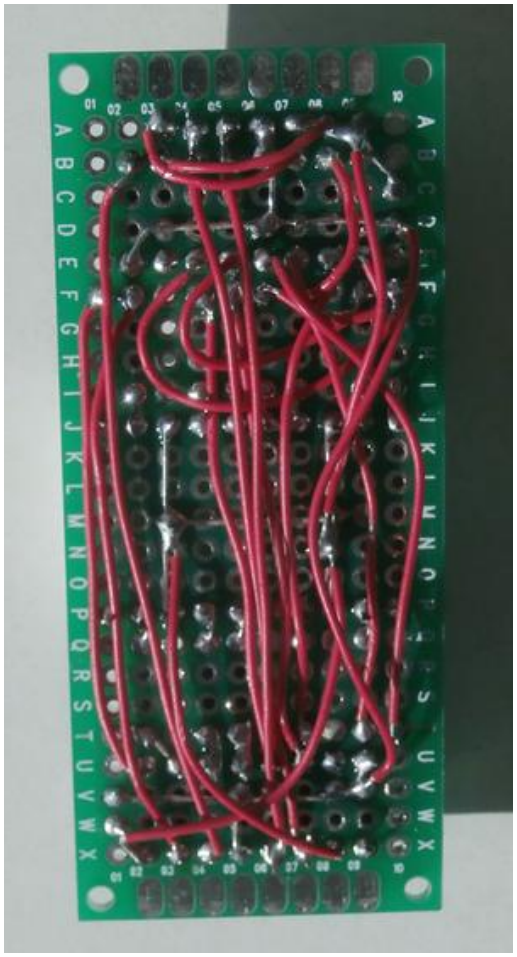
Kétszámjegyű kijelző

A shift regisztereket sorba köthetjük, így többszámjegyű kijelzőt építhetünk.



Hevenyészett kivitel 3x7 cm-es próbapanelon





Előny:
Nem kell multiplex vezérlés

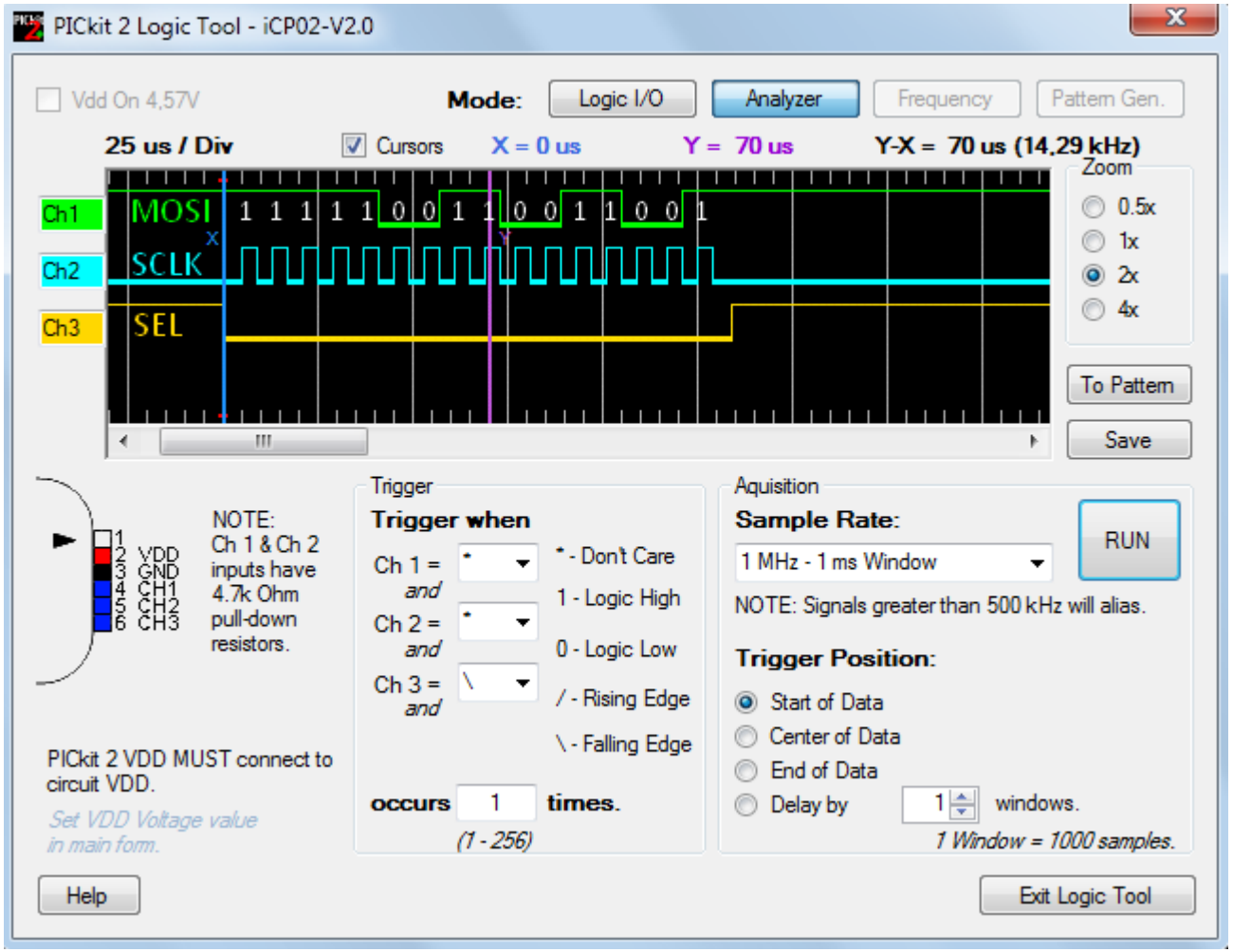
Hátrány:
Sok ellenállást kell beépíteni (nagyobb helyigény)

Program8_1

```
#include "MKL25Z4.h"

void SPI0_init(void) {
    SIM->SCGC5 |= 0x1000;          // Port D engedélyezése
    PORTD->PCR[1] = 0x200;         // PTD1 legyen SPI SCK */
    PORTD->PCR[2] = 0x200;         // PTD2 legyen SPI MOSI */
    PORTD->PCR[0] = 0x100;         // PTD0 legyen GPIO módban
    PTD->PDDR |= 0x01;             // PTD0 kimenet legyen (SPI SS)
    PTD->PSOR = 0x01;              // Kezdetben '1' legyen

    SIM->SCGC4 |= 0x400000;        // Az SPI0 modul engedélyezése
    SPI0->C1 = 0x10;               // SPI letiltása, master mód
    SPI0->C1 |= 0x01;              // LSBFE = 1 (elöször LSB-t küldjük)
    SPI0->C2 = 0;                  // Alapértelmezett beállítások
    SPI0->BR = 0x54;               // Baud rate = 125 kHz (/6 és /32 osztók)
    SPI0->C1 |= 0x40;              // Az SPI modul engedélyezése
}

void SPI0_write(unsigned char data) {
    volatile char dummy;
    while(!(SPI0->S & 0x20)) { }    /* wait until tx ready */
    SPI0->D = data;                 /* send data byte */
    while(!(SPI0->S & 0x80)) { }    /* wait until tx complete */
    dummy = SPI0->D;                /* clear SPRF */
}
```

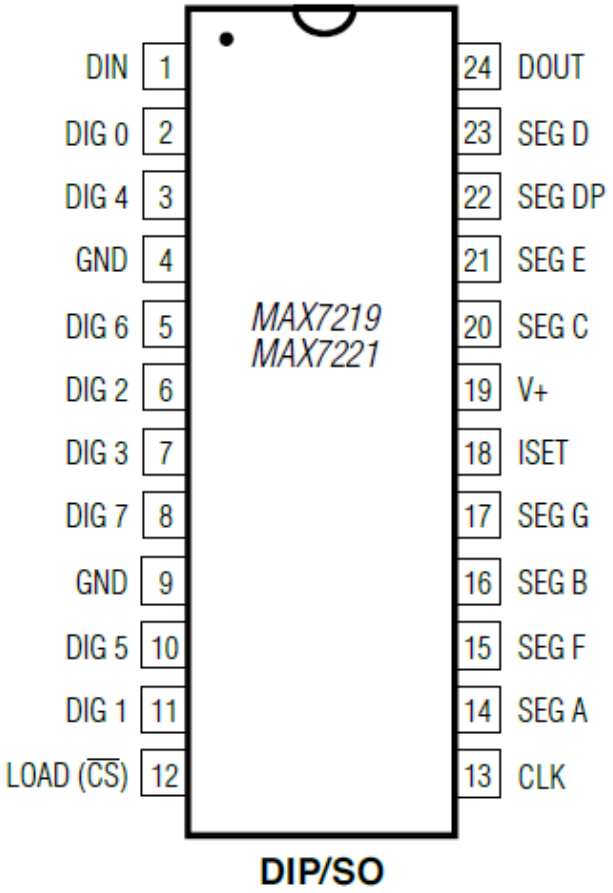
```
const unsigned char digit [10] = {
  0xFC,  // 0b11111100 - 0
  0x60,  // 0b01100000 - 1
  0xDA,  // 0b11011010 - 2
  0xF2,  // 0b11110010 - 3
  0x66,  // 0b01100110 - 4
  0xB6,  // 0b10110110 - 5
  0xBE,  // 0b10111110 - 6
  0xE0,  // 0b11100000 - 7
  0xFE,  // 0b11111110 - 8
  0xF6}; // 0b11110110 - 9

int main(void) {
    unsigned char n, d0, d1;
    SPI0_init();                // Az SPI0 modul konfigurálása
    while(1) {
        for(n=0; n<100; n++) {
            d1 = ~digit[n/10];  // Elso számjegy szegmensei
            d0 = ~digit[n%10];  // Második számjegy szegmensei
            PTD->PCOR = 1;      // SS aktiválása
            SPI0_write(d0);     // Egyesek kiküldése
            SPI0_write(d1);     // Tízesek kiküldése
            PTD->PSOR = 1;      // SS deaktiválása
            delayMs(500);
        }
    }
}
```

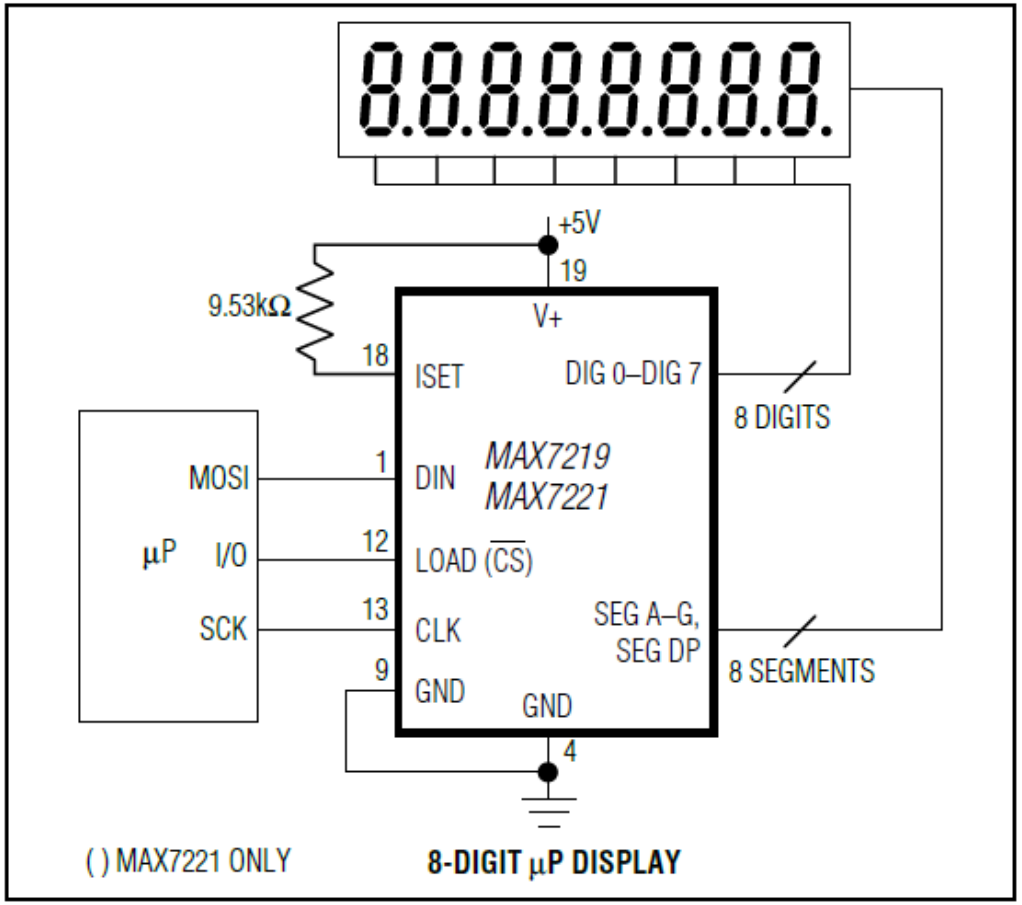
A jelalak vizsgálata logikai analizátorral



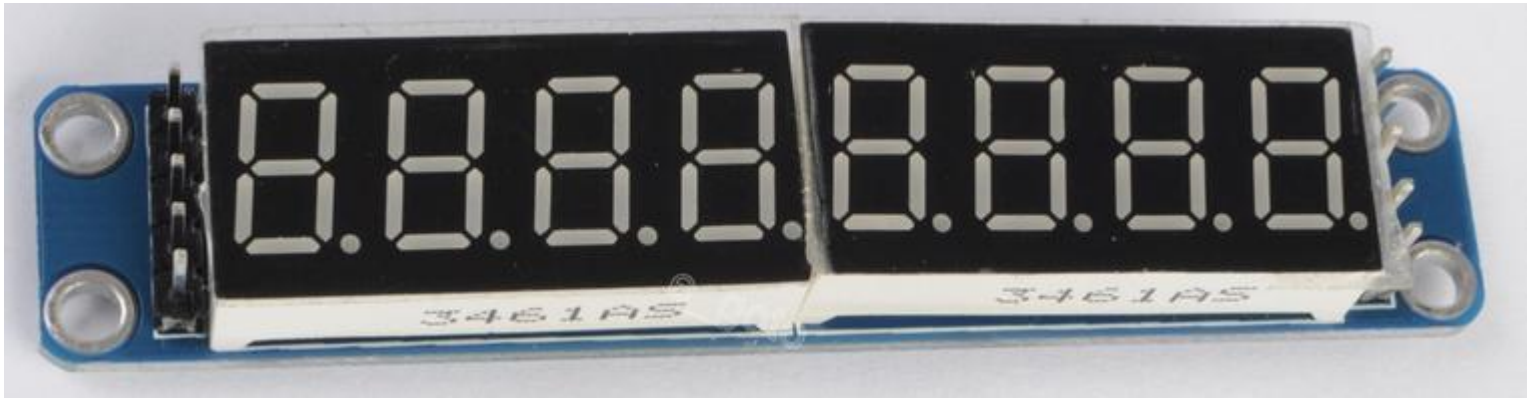
Program8_2: kijelző vezérlése MAX7219 IC-vel

Egy Maxim Integrated **MAX7219** LED vezérlővel IC-vel ellátott, 8-digites, hétszegmenses számkijelző modult használunk, ami SPI illesztőfelülettel rendelkezik. Beépített áramkorlátozással rendelkezik, s a fényerő 32 lépésben programozottan is változtatható. A SEG A – SEG DP kimenetek áramforrások, a DIG 0 – DIG 7 kimenetek áramnyelők.





Készen kapható kijelző modul



Technikai részletek a konfiguráláshoz

MAX7219 regisztertérkép

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

Nem használt
DP a b c d e f g

0: no decode 1: decode

0 – 0xF
0 – 7
0: shutdown 1: normal mode

1: test mode 0: normal mode

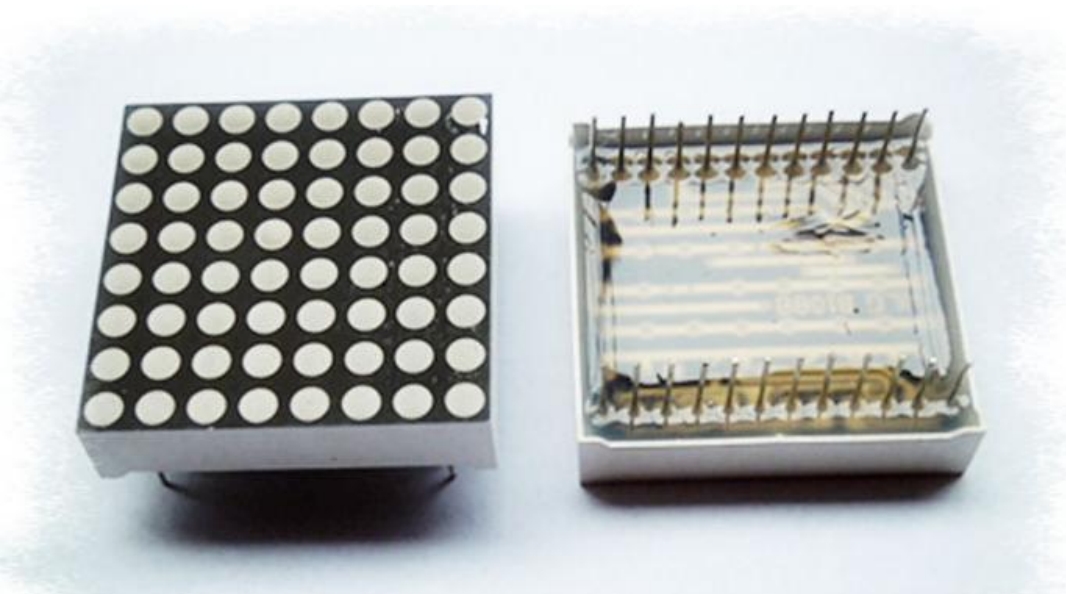
Program8_2 listája (részletek)

```
#include "MKL25Z4.h„

int main(void) {
  unsigned char i;
  SPI0_init();                    // SPI0 konfigurálása
  max7219_write(DECODE, 0xFF); // dekódolás 8 számjegyre
  max7219_write(SCANLIMIT, 7); // pásztázás 8 jegyre
  max7219_write(INTENSITY, 8); // Kitöltési arány = 17/32
  max7219_write(TESTMODE, 1);  // Teszt mód engedélyezése
  max7219_write(SHUTDOWN, 1);  // Megjelenítés engedélyezése
  for(i=1; i<9; i++) max7219_write(i,0x0F);   // Blank karakter
  delayMs(1000);
  max7219_write(TESTMODE, 0);  // Teszt mód letiltása
  delayMs(1000);
  while(1) {
    for(i=1; i<9; i++) {          // Számjegyek kiírása
      max7219_write(i,i-1);
      delayMs(500);
    }
    delayMs(1000);
    for(i=1; i<9; i++) {          // Számjegyek törlése
      max7219_write(i,0x0F);    // Blank karakter
      delayMs(500);
    }
  }
}
```

```
#define NO_OP       0
#define DECODE      9
#define INTENSITY  10
#define SCANLIMIT  11
#define SHUTDOWN   12
#define TESTMODE   15
```

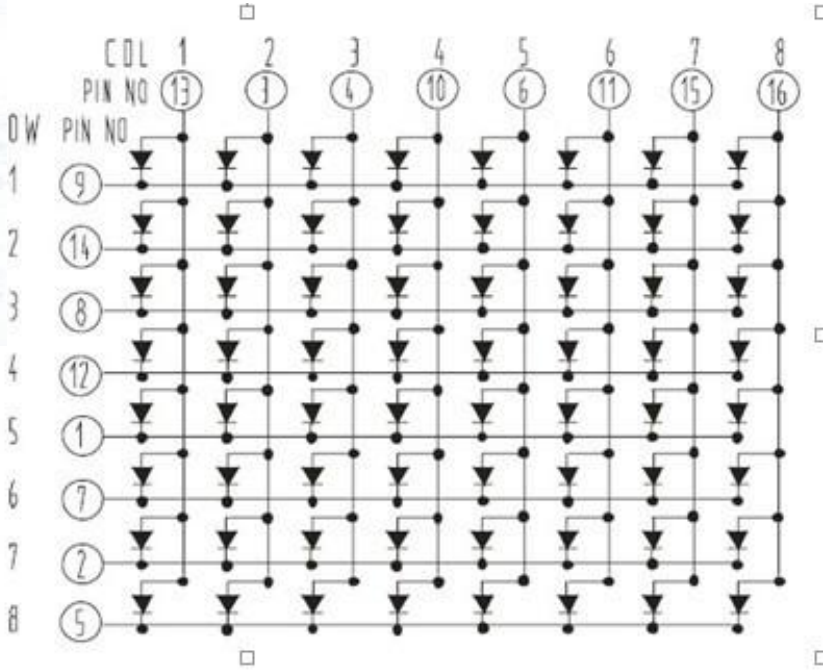
```
void max7219_write(unsigned char command, unsigned char data) {
    volatile char dummy;
    PTD->PCOR = 1;                    // SS aktiválás
    while(!(SPI0->S & 0x20)) { }      // TX kész jelre vár
    SPI0->D = command;                // Parancs küldése
    while(!(SPI0->S & 0x80)) { }      // Átvitel végére vár
    dummy = SPI0->D;                  // SPRF törlése
    while(!(SPI0->S & 0x20)) { }      // TX kész jelre vár
    SPI0->D = data;                   // Adat küldése
    while(!(SPI0->S & 0x80)) { }      // Átvitel végére vár
    dummy = SPI0->D;                  // SPRF törlése
    PTD->PSOR = 1;                    // SS deaktiválása
}

void SPI0_init(void) {
    SIM->SCGC5 |= 0x1000;          // Port D engedélyezése
    PORTD->PCR[1] = 0x200;         // PTD1 legyen SPI SCK */
    PORTD->PCR[2] = 0x200;         // PTD2 legyen SPI MOSI */
    PORTD->PCR[0] = 0x100;         // PTD0 legyen GPIO módban
    PTD->PDDR |= 0x01;             // PTD0 kimenet legyen (SPI SS)
    PTD->PSOR = 0x01;              // Kezdetben '1' legyen
    SIM->SCGC4 |= 0x400000;        // Az SPI0 modul engedélyezése
    SPI0->C1 = 0x10;               // SPI letiltása, master mód, MSB elsőként
    SPI0->C2 = 0;                  // Alapértelmezett beállítások
    SPI0->BR = 0x51;               // Baud rate = 1 MHz (/6 és /4 osztók)
    SPI0->C1 |= 0x40;              // Az SPI modul engedélyezése
}
```

Program8_3: LED 8x8 mátrix vezérlése MAX7219 IC-vel



3 mm-es piros LED-ek
8x8 mátrixba szervezve

1088AS vagy **M1388AR** típusnál a sorkiválasztó vonal a közös katód

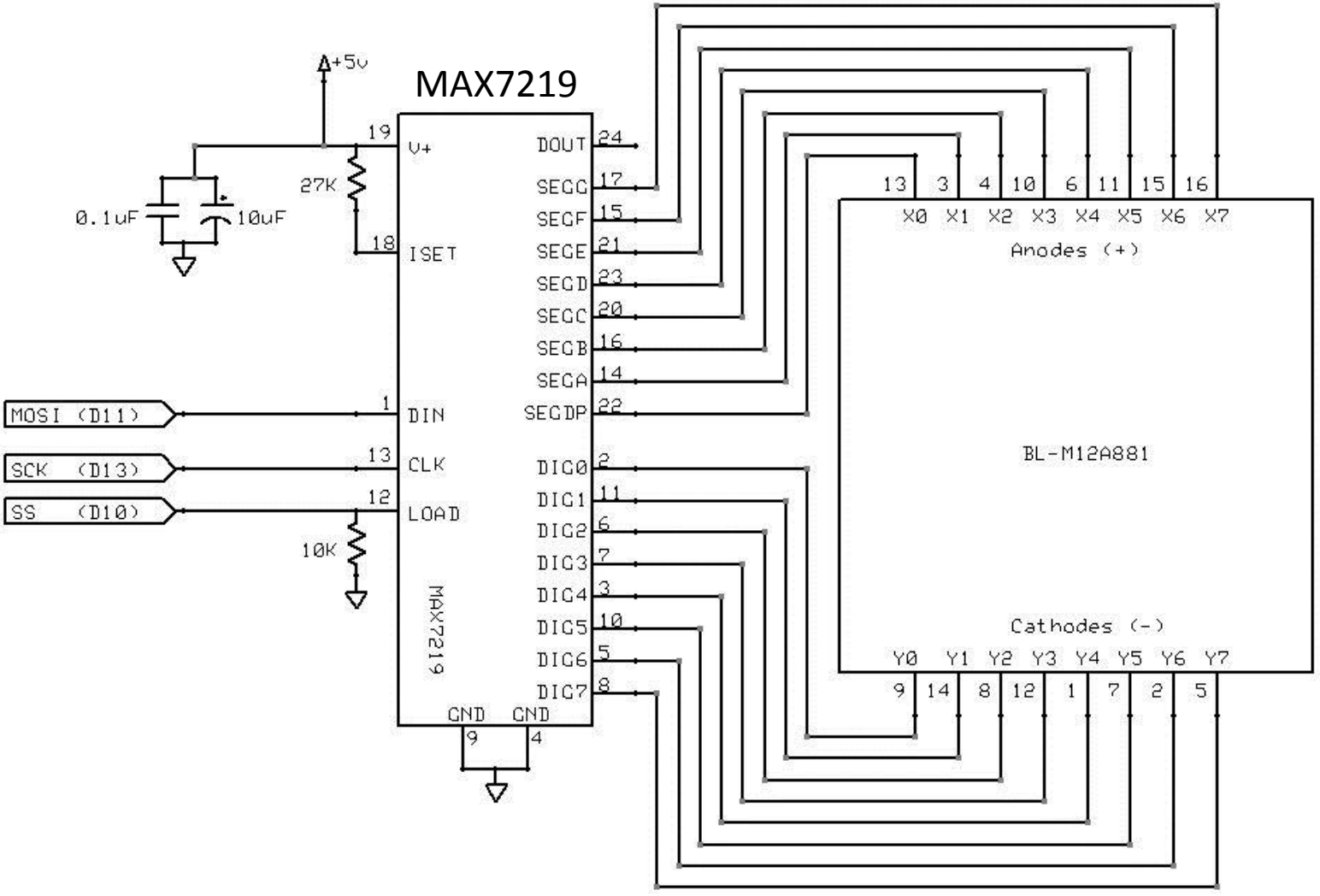


Multiplex kijelzés, egyidejűleg legfeljebb egy sor, vagy egy oszlop lehet aktív.

Kényelmes meghajtás:

- 1 db **MAX7219**, vagy
- 2 db **74HC595** (+ meghajtó +áramkorlátozás)
- 1 db **MCP23S017** (+ meghajtó +áramkorlátozás)

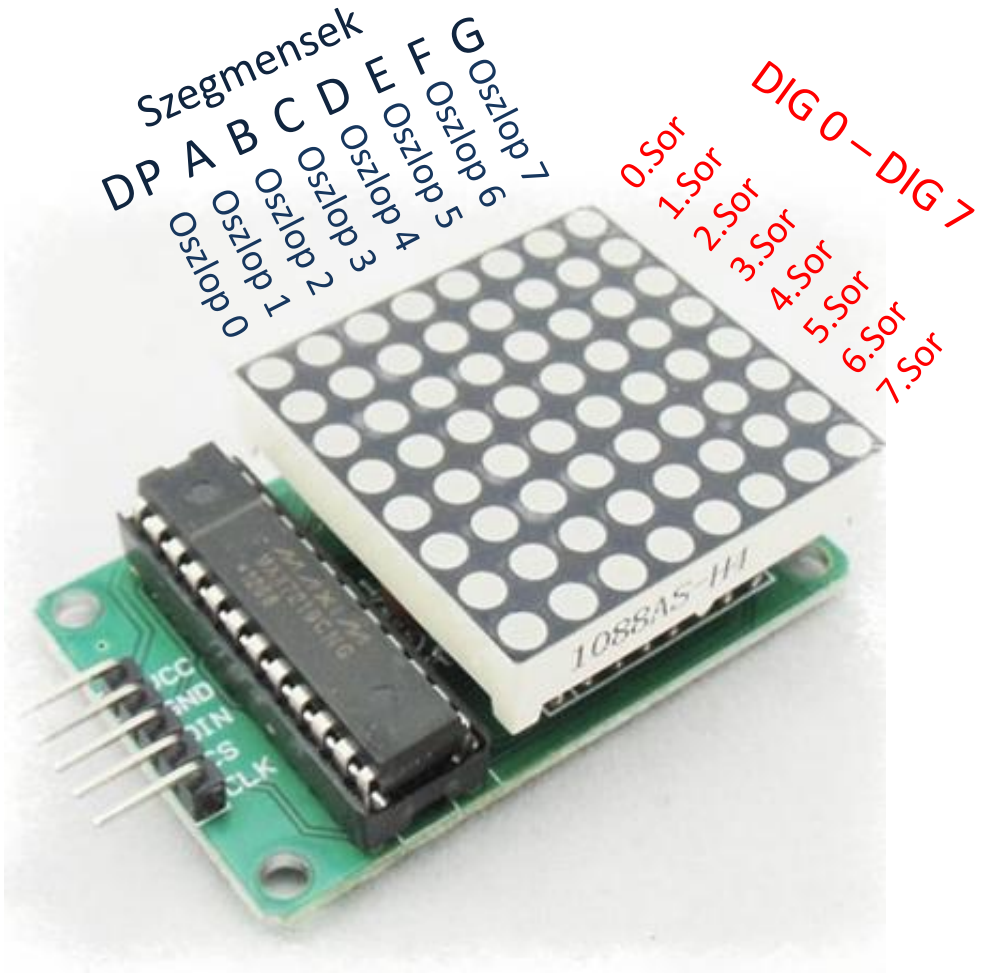
8x8-s LED mátrix vezérlése



Komplett kijelző modul (pl. Ebay.com)

- 8x8 LED mátrix
- MAX7219 vezérlő
- Felfűzhető kivitel
- Tápellátás: 3,5 – 5 V

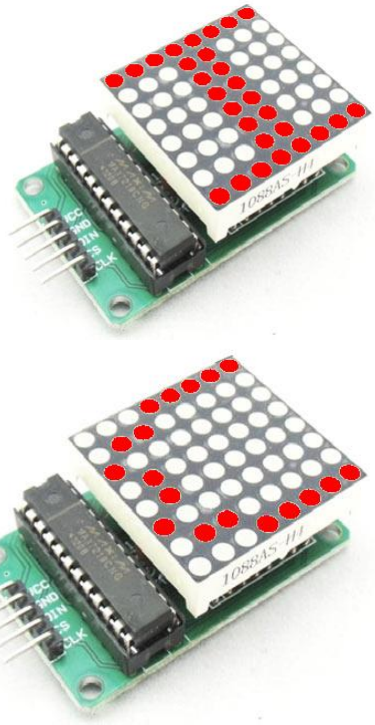
Bemenetek	**Kimenetek**
1 VCC	1 VCC
2 GND	2 GND
3 **DIN**	3 **DOUT**
4 CS	4 CS
5 CLK	5 CLK



Program8_3 lista (részletek)

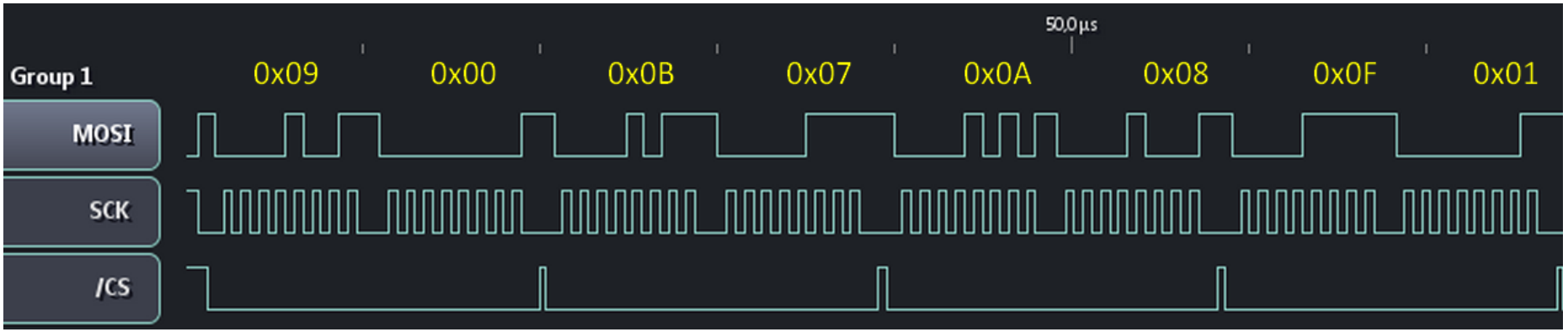
```
#include "MKL25Z4.h"
const char minta1[]= {0xFF,0x18,0x18,0x18,0x18,0x18,0x18,0xFF};  //H
const char minta2[]= {0x1F,0x60,0x80,0x40,0x40,0x80,0x60,0x1F};  //W

int main(void) {
    unsigned char i;
    SPI0_init();                    // SPI0 konfigurálása
    max7219_write(DECODE, 0);       // Nincs dekódolás
    max7219_write(SCANLIMIT, 7); // pásztázás 8 sorra/oszlopra
    max7219_write(INTENSITY, 8); // Kitöltési arány = 17/32
    max7219_write(TESTMODE, 1);  // Teszt mód engedélyezése
    max7219_write(SHUTDOWN, 1);  // Megjelenítés engedélyezése
    for(i=1; i<9; i++) {
        max7219_write(i,0);         // képpontok törlése
    }
    delayMs(1000);
    max7219_write(TESTMODE, 0);  // Teszt mód letiltása
    delayMs(1000);
    while(1) {
      for(i=1; i<9; i++) max7219_write(i,minta1[i-1]);
      delayMs(1000);
      for(i=1; i<9; i++) max7219_write(i,minta2[i-1]);
      delayMs(1000);
    }
}
```

Minden szegmens állapotát magunk állítjuk be!



Az inicializáló parancsok vizsgálata

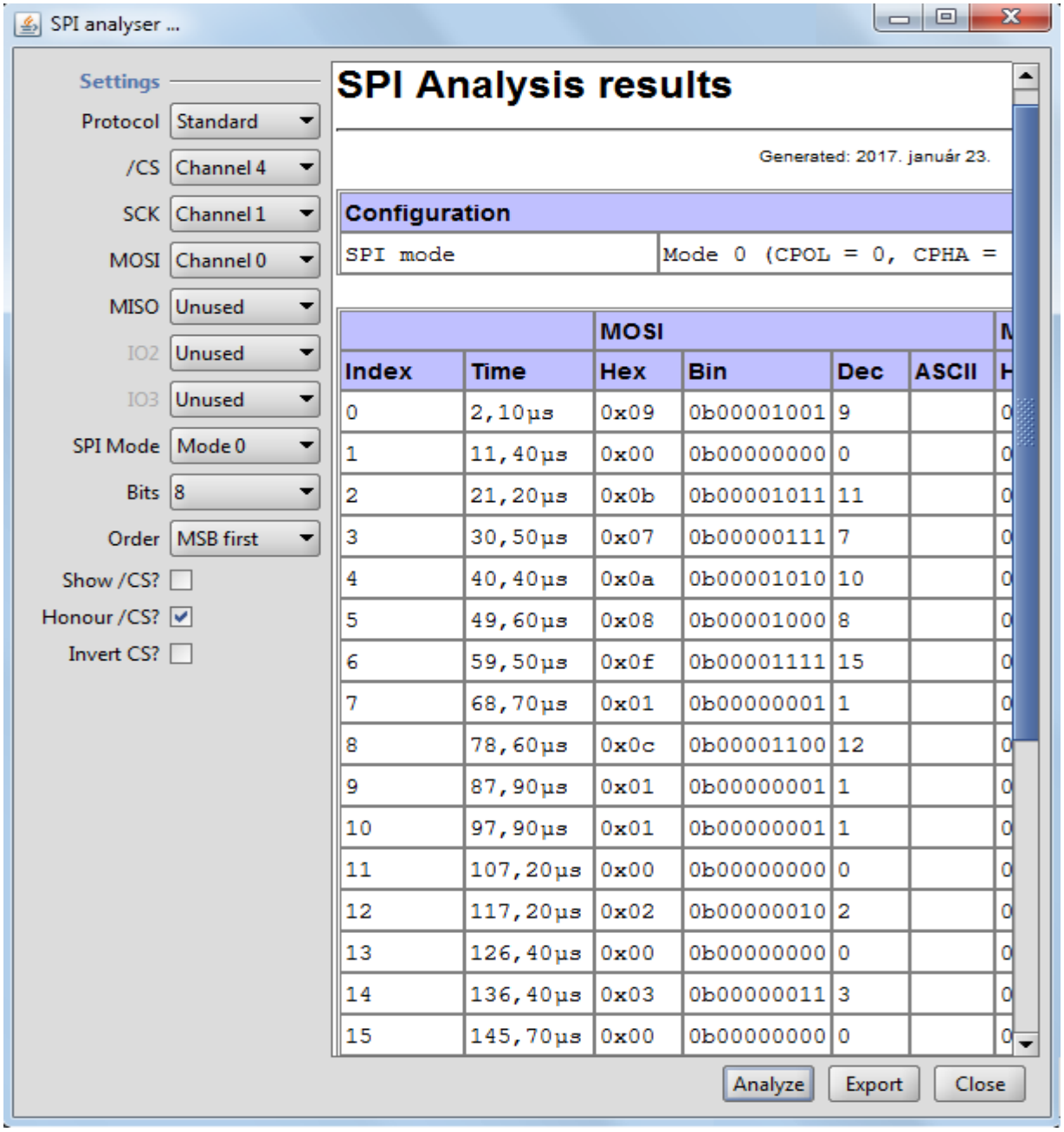
Hardver: Texas Instruments Stellaris Launchpad (max. 8 csatorna az RB0 – RB7 bemeneteken)
Szoftver: SLLogicLogger firmware (10 MHz mintavételezés) + Open Bench Logic Sniffer (PC alkalmazás)



Az ábrán a MAX7219 konfigurálásának első parancsai láthatók. Bitsorrend: MSB first

```
max7219_write(DECODE, 0);     //Nincs dekódolás
max7219_write(SCANLIMIT, 7); //pásztázás 8 oszlopra
max7219_write(INTENSITY, 8); //Kitöltés: 17/32
max7219_write(TESTMODE, 1);   //Teszt mód be
max7219_write(SHUTDOWN, 1);   //Megjelenítés be
```

```
#define DECODE     9
#define INTENSITY 10 //0x0A
#define SCANLIMIT 11 //0x0B
#define SHUTDOWN  12 //0x0C
#define TESTMODE  15 //0x0F
```

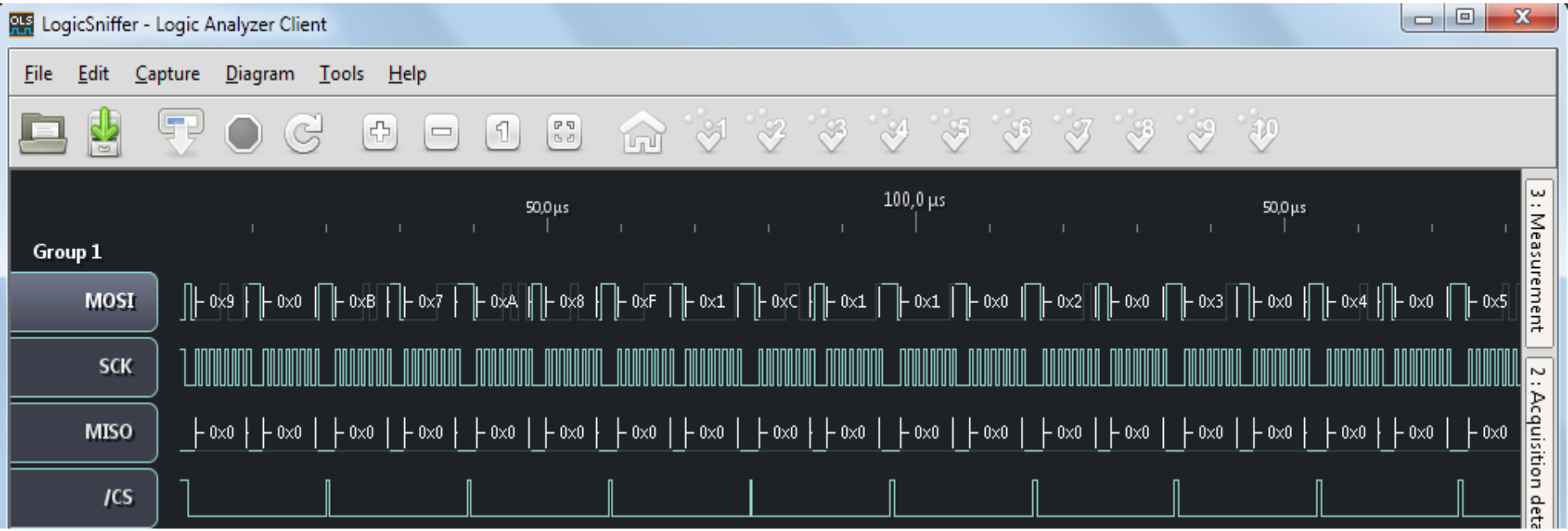


Az Open Bench Logic Sniffer alkalmazás sokféle kommunikációs protokoll jeleinek értelmezésére is képes (UART, I2C, SPI stb).

Itt az SPI protokoll szerinti értelmezést kértük, ennek eredménye látható a táblázatban.

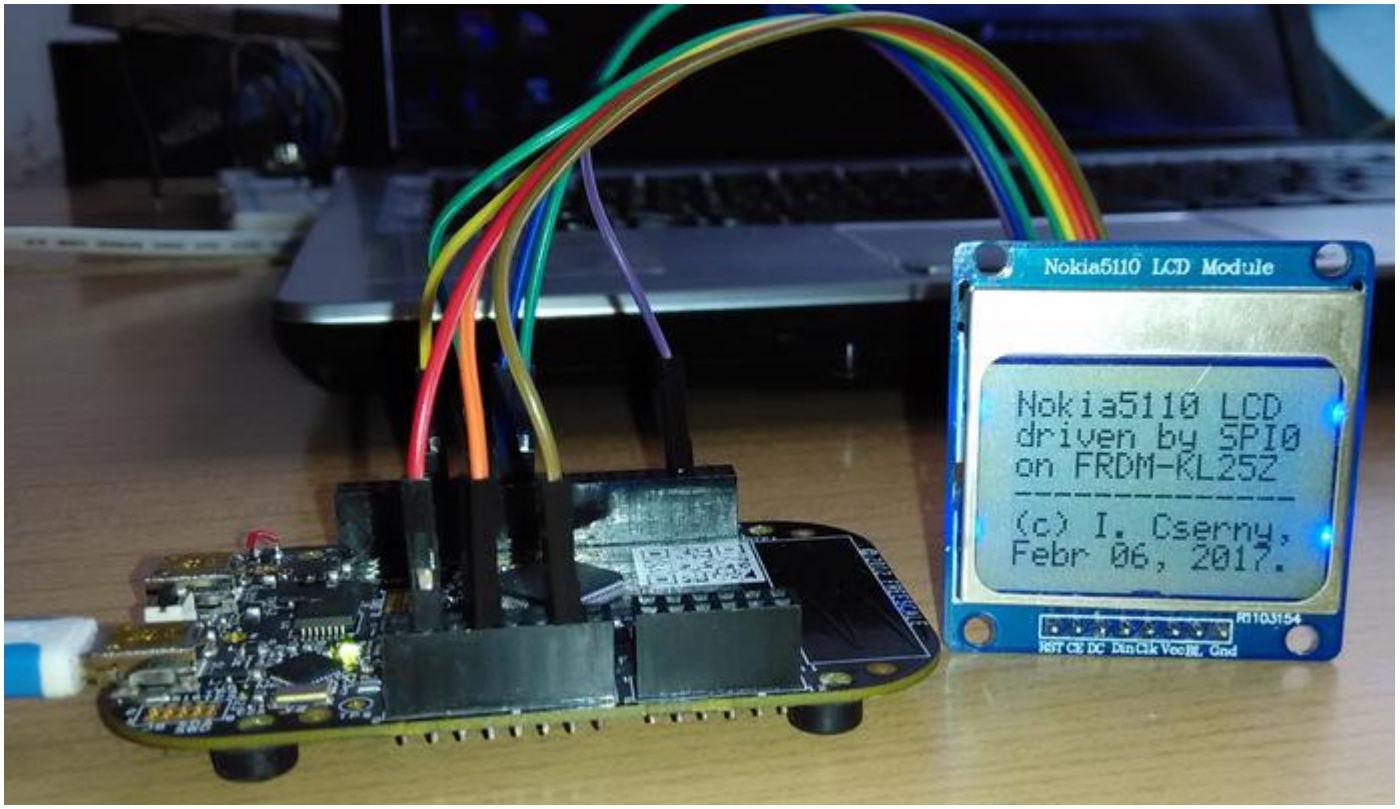
Az SPI tranzakciók dekódolása

A képen az Open Bench Logic Sniffer alkalmazás által értelmezett SPI tranzakciók láthatók, melyekben a MAX7219 konfigurálására és az adatregiszterek nullázására ismerhetünk rá.



Nokia 5110 kijelző vezérlése

Nokia 5110 kijelző: monkróm reflexiós LCD, LED oldalvilágítással
Vezérlő: **PCD8544** (SPI interfész)
Felbontás: **84 x 48 képpont**
Kivezetések: **VCC, GND, LED, SCLK, DIN, D/C, CE, RST**
Tápfeszültség: **3,3 V – 5 V**



Program8_4

A program felváltva egy bitképet, illetve egy szöveges képernyőt mutat.
A bitképet az **nxp_logo.h** állomány tartalmazza hexadecimális adatsorként.
A betűképeket az **english_6x8_pixel.h** állomány definiálja hexadecimális adatsorként.

```
#include "MKL25Z4.h"
#include "english_6x8_pixel.h"
#include "nxp_logo.h„

int main(void) {
    SPI0_init();                                  // SPI0 konfigurálása
    LCD_init();
    while(1) {
      LCD_write_logo(nxp_logo);                   // Bitkép megjelenítés
      delayMs(5000);                              // 5 s várakozás
      LCD_clear();                                // Képernyö törlése
      LCD_write_string(0,0,"Nokia5110 LCD ");
      LCD_write_string(0,1,"driven by SPI0");
      LCD_write_string(0,2,"on FRDM-KL25Z ");
      LCD_write_string(0,3,"--------------");
      LCD_write_string(0,4,"(c) I. Cserny,");
      LCD_write_string(0,5,"Febr 06, 2017.");
      delayMs(5000);
    }
}
```

```
void SPI0_init(void) {
    SIM->SCGC5 |= 0x1000;          // Port D engedélyezése
    PORTD->PCR[0] = 0x100;         // PTD0 legyen GPIO módban
    PORTD->PCR[1] = 0x200;         // PTD1 legyen SPI SCK */
    PORTD->PCR[2] = 0x200;         // PTD2 legyen SPI MOSI */
    PORTD->PCR[4] = 0x100;         // PTD4 legyen GPIO módban
    PORTD->PCR[5] = 0x100;         // PTD5 legyen GPIO módban
    PTD->PDDR |= 0x31;             // PTD0, PTD4, PTD5 legyen kimenet
    PTD->PSOR = 0x11;              // PTD0 és PTD4 '1' legyen
    SIM->SCGC4 |= 0x400000;        // Az SPI0 modul engedélyezése
    SPI0->C1 = 0x10;               // SPI letiltása, master mód, MSB elöször
    SPI0->C2 = 0;                  // Alapértelmezett beállítások
    SPI0->BR = 0x51;               // Baud rate = 1 MHz (/6 és /4 osztók)
    SPI0->C1 |= 0x40;              // Az SPI modul engedélyezése
}

void LCD_init(void) {
    PTD->PCOR = 0x10;              // LCD_RST lehúzása
    delayMs(100);
    PTD->PSOR = 0x10;              // LCD_RST felhúzása
    delayMs(100);
    LCD_write_byte (0x21, 0);      // use the extended command set the LCD mode
    LCD_write_byte (0xc8, 0);      // set the bias voltage
    LCD_write_byte (0x06, 0);      // temperature correction
    LCD_write_byte (0x13, 0);      // 1:48
    LCD_write_byte (0x20, 0);      // use basic commands
    LCD_clear ();                  // clear the screen
    LCD_write_byte (0x0c, 0);      // set display mode, the normal display
  }
```

```
void LCD_write_byte(unsigned char data, unsigned char dc) {
    unsigned char dummy;
    PTD->PCOR = 1;                  // SS aktiválás
    if(dc) { PTD->PSOR = 0x20;}     // D/C = 1
    else PTD->PCOR = 0x20;          // D/C = 0
    while(!(SPI0->S & 0x20));       // TX kész jelre vár
    SPI0->D = data;                 // Adat küldése
    while(!(SPI0->S & 0x80));       // Átvitel végére vár
    dummy = SPI0->D;                // SPRF törlése
    PTD->PSOR = 1;                  // SS deaktiválása
}

void LCD_write_logo(unsigned const char *ptr) {
    unsigned int ctr = 0;
    while(ctr++ < 504) {
      LCD_write_byte(*ptr++,1);
    }
}

void LCD_clear(void) {
    unsigned int i;
    LCD_write_byte(0x0c, 0);
    LCD_write_byte(0x80, 0);
    for (i=0; i<504; i++)
      LCD_write_byte(0, 1);
}
```

```
// LCD_set_XY: kurzor beállítása megadott helyre
// x: karakterhely száma (0-83)
// y: sor száma (0-5)
void LCD_set_XY(unsigned char X, unsigned char Y)
  {
    LCD_write_byte(0x40 | Y, 0); // column
    LCD_write_byte(0x80 | X, 0); // row
  }

void LCD_write_char(unsigned char c) {
    unsigned char line;
    for (line=0; line<6; line++)
      LCD_write_byte(font6x8[c-32][line], 1);
}

void LCD_write_string(unsigned char x,unsigned char y,char *s) {
    LCD_set_XY(x,y);
    while (*s) LCD_write_char(*s++);
}

//---------------------------------------
// Késleltető függvény 48 MHz órajelhez
//---------------------------------------
void delayMs(int n) {
    int i, j;
    for(i = 0 ; i < n; i++)
        for (j = 0; j < 8010; j++);
}
```

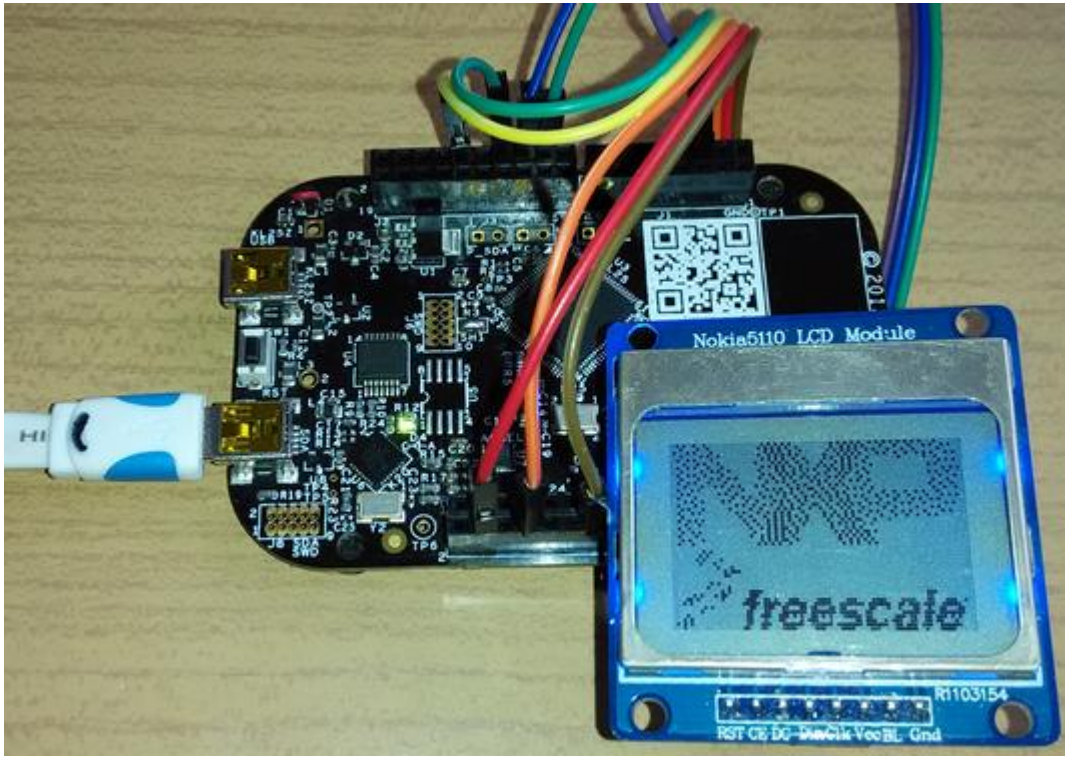
Összekötési vázlat

Hardver követelmények:

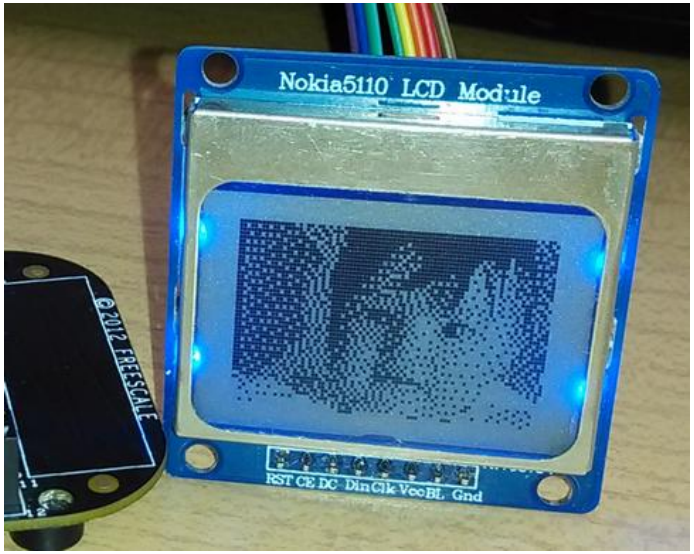
- FRDM-KL25Z kártya
- Nokia5110 kijelző

Arduino kompatibilis lábkiosztást használunk:

```
D13  (PTD1): SPI SCK   ===>  LCD SCLK
D12  (PTD3): SPI MISO  nem használjuk
D11  (PTD2): SPI MOSI  ===>  LCD DIN
D10  (PTD0): SPI SS    ===>  LCD CE
D9   (PTD5)            ===>  LCD D/C (vagy RS)
D2   (PTD4)            ===>  LCD RST (RESET)
```

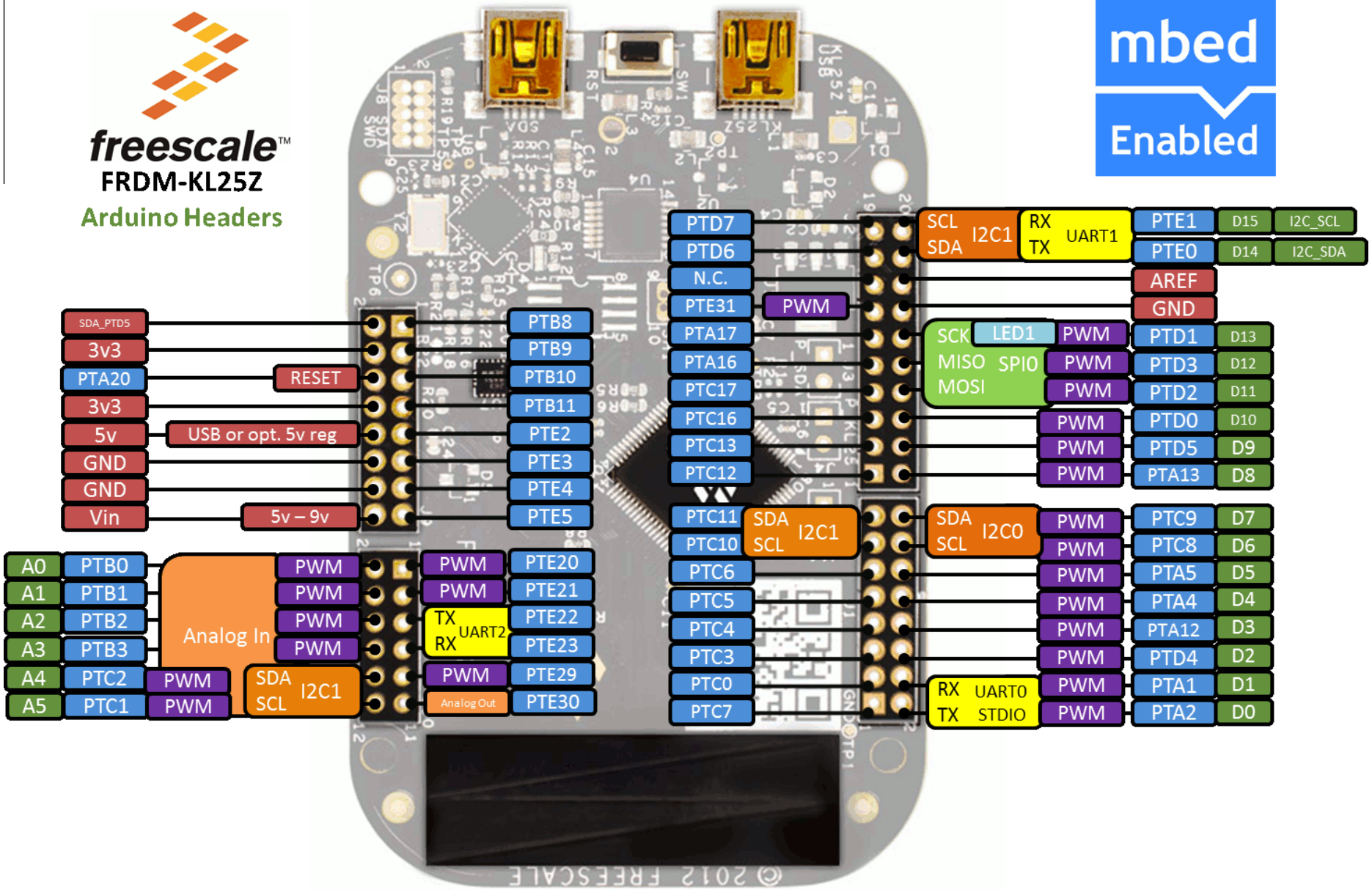


Kép adattá konvertálás: LCD Assistant
en.radzio.dxp.pl/bitmap_converter/



freescale™
FRDM-KL25Z
Arduino Headers

mbed Enabled



freescale™

FRDM-KL25Z

Additional Peripherals